REMARKS

Claims 1 to 9 are pending. Claims 1, 8 and 9 are amended to emphasize that programming properties of programming objects are reflected through graphical properties of graphical elements. No new issues are raised by these amendments since they merely amplify or emphasize the nature of the invention.

Claims 1 to 9 were rejected under 35 U.S.C. §102(e) as being anticipated by U.S. Patent No. 5,883,639 to Walton et al. This rejection is again traversed for the reason that Walton et al. do not show the claimed invention.

The claimed invention is directed to a visual programming language in which programming objects in memory are represented as graphical objects on a display and wherein when program properties of the programming objects change, changes are induced in the graphical properties of the visual representation of the object on the display. The invention is used in static program analysis which aims at determining properties of the behavior of a program without actually executing it. Static analysis is founded on the theory of abstract interpretation for proving the correctness of analyses with respect to the semantics of a programming language. An analysis can often be divided into two phases:

1. A first phase in which a program is translated into a system of equations or constraints over a partial order of program properties. The solutions to the system represent correct information about the particular property which is being analyzed.
2. A second resolution phase in which an actual solution is calculated. This phase can rely on either general iterative work-set algorithms or on more domain-specific symbolic resolution techniques.

The disclosed and claimed invention is directed to a specific aid in such static program analysis.

Figure 3 shows a block diagram of a visual programming language of the present invention. This diagram illustrates programming object *state reflection*. A display 400 depicts visual representations 440, 450 and 460 of the programming objects 470, 480 and 490 in memory 410. The visual programming language

executable 430 maintains the relationships between the visual representations and the programming objects. In the course of programming with the programming objects 470, 480 and 490, the states of these programming objects may change. For example, the data they represent may not be initialized, or they may be set to some default values. Through program analysis or other means, general characteristics may be determined, for example, that their values may be within a certain set of values or ranges. This all depends upon the nature of the variables, and the values they may be assigned. When for a given programming object, at any point in programming, characteristics such as these are determined, it is said that the programming object's programming state is determined. These programming states can induce changes in the graphical properties of the visual representation of the object. By way of illustration, if a programming object is known to have been set to some default value, the color of some related graphical element might be set to red. This process is known as *state reflection*, and this process is the essence of the claimed invention.

In contrast, Walton et al. disclose a visual software engineering system which uses a concept of defining both input to and output from graphical objects in an object-oriented system by providing examples of what the user desires the graphical object to do. This technique is referred to by Walton et al. as "animation by example". In this process, the user creates a user interface by drawing the user interface with a graphics editor and then defining the output behavior (i.e., graphics manipulation) of the user interface components by showing each state or frame as an animation. This is accomplished by changing the object using a graphic editor function such as move or rotate and *storing each of the frames with the object as a behavior state*. Just as with defining the output, the input is defined by giving the graphic object an example of what type of input to look for, and once it finds that input, it tells the object which frame to output or change to. *Application code can then drive the animation or read the input by accessing the frame numbers assigned to each of the example frames*. (See Abstract of Walton et al.)

Thus, Walton et al. disclose a visual software engineering system for

providing a means of *creating user interfaces* to products under development *without the need of a programming language*. On a graphical interface, the user *designs* a user interface, including visual artifacts such as buttons, check boxes, etc. These visual entities are managed by "graphical objects". A graphical object is not a generic programming object, but rather is an object subject to enablement. For a given set of visual objects, the user sets buttons, moves dials, etc., and when a desired visual state is found, the user "snapshots" that visual state, gives it a name, and all this state information plus name is stored as a "frame" with the graphical object. The user does this any number of times. Similarly, input can be defined on the graphical object "by example", which the graphical object later detects as a trigger to set up a specific state based on a specific user-specified "frame".

To summarize, the Walton et al. graphical object is/has the following:

1) A well defined object for managing graphical interfaces.
2) Maps to graphical representations.
3) Stores "frames" or visual states of a user interface.
4) Maps user input (mouse, etc.) to frames.

Finally, with reference to column 8, lines 54–56, Walton et al. state "The resulting objects are then stored as objects in an object-oriented database system and connected to other objects or user code . . ." This is how the graphical object is related to user code. Essentially, the user code has to "connect to" or invoke the graphical objects. The means are not precise, and do not need to be. What is important is that the graphical objects are not user code defined objects; rather, they are service objects to the user application program.

A distinguishing feature of the disclosed invention is found in the specification on page 9, beginning at line 16, wherein implicitly the programming objects (variables, objects) are part of *an existing or developing* application program. What this means is that the state or program property of the programming object, say at a line in the program, is derived from some kind of code analysis (data flow, control flow, pointer, etc.). State is very broadly put, but includes things like initialization state (set/unset/ "set to 3") or known ranges (x is

between 3 and 9). The claims have been amended to use the term "program property" rather than "state" to better clarify what is being analyzed. The present invention describes the mechanism for displaying that change in a program property of a programming element..

In contrast to Walton et al.,

1) The claimed object refers to user application objects (variables, etc.), not the specific graphical object as described in Walton et al;
2) The program property of the programming object is derived from a program context and the state is displayed. This is the opposite direction from Walton et al., who defines state and stores it, and maps actions. The former is a reflective paradigm, while the latter is more of a constructive paradigm.

These distinctions have been clearly set out in the claims and are now emphasized by the amendments to the independent claims. Specifically, claim 1 recites a "computer implemented method of visual representation of programming objects as graphical elements, *wherein program properties of said programming objects are reflected through graphical properties of graphical elements*" (emphasis added). The method comprises three steps. The first step is "*detecting a change in a program property of a programming object* in visual representation and shown visually on a display device as one or more graphical elements, wherein graphical elements represent the programming object and *program properties of programming objects are reflected through graphical element properties*" (emphasis added). The second step is "*determining graphical aspect changes that apply to graphical elements of the programming object appropriate for the change in a program property of the programming object*" (emphasis added). The third step "applying the graphical aspect changes to corresponding graphical elements, wherein the graphical aspect changes include changes in color, position and size." This last step is the process of *state reflection* "wherein programming properties of programming objects are reflected through graphical properties of graphical elements", as recited in the preamble.

As explained above, Walton et al. do not contemplate the process of *state reflection* as a result of detecting a change in state of a data element representing a programming object.

Claim 8 recites an "apparatus for visual representation of programming objects as graphical elements". This apparatus comprises "a data processing system comprising a display device, an interactive device, as in a keyboard, a pointing device, a storage device, and a data processor", a "memory coupled to the data processor via a bidirectional bus, wherein the memory includes a first memory section for at least one program and a second memory section for data", "computer code . . .", and "means for displaying a visual representation of a plurality of graphical elements on the display device. . ." The computer code comprises "a visual programming language, wherein the computer code is stored in the first memory section, and the computer code *detects a change in a program property* of a programming object, *determines graphical aspect changes that apply to graphical elements* which represent the programming object, and *applies graphical aspect changes to said visual representation of said programming object which represents the change of the program property of the programming object*" (emphasis added). The "displayed graphical elements represent programming objects and program properties of programming objects are reflected through displayed graphical element properties." Thus, claim 8 similarly defines over Walton et al.

Claim 9 is similar to claim 1, but is directed to a machine readable medium containing code for performing the method of claim 1. The language of claim 9 is otherwise similar to that of claim 1.

In view of the foregoing, it is respectfully requested that the application be reconsidered, that claims 1 to 9 be allowed, and that the application be passed to issue. In the alternative, it is requested that this amendment be entered for purposes of appeal.

Should the Examiner find the application to be other than in condition for allowance, the Examiner is requested to contact the undersigned at the local telephone number listed below to discuss any other changes deemed necessary in a telephonic or personal interview.

A provisional petition is hereby made for any extension of time necessary for the continued pendency during the life of this application. Please charge any fees for such provisional petition and any deficiencies in fees and credit any overpayment of fees to Attorney's Deposit Account No. 50-2041.

Respectfully submitted,

C. Lamont Whitham
Reg. No. 22,424

Whitham, Curtis & Christofferson, P.C.
11491 Sunset Hills Road, Suite 340
Reston, VA 20190
Tel. (703) 787-9400
Fax. (703) 787-7557
Customer No.: 30743